A function whose order of growth is larger than any polynomial function but smaller than any exponential function can be achieved using logarithmic-exponential functions. One such example is the **super-polynomial function**:

$$f(n) = n^n$$

However, to be more precise within the bounds specified (i.e., between polynomial and exponential growth), the **super-polynomial but sub-exponential function** would be:

$$f(n) = n^{\log n}$$

Explanation:

- **Polynomial functions** have the form n^k for some constant k.
- **Exponential functions** have the form a^n for some constant $a > 1$.

The function $f(n) = n^{\log n}$ grows faster than any polynomial function n^k but slower than any exponential function a^n.

Growth Rate Analysis:

1. **Compare with Polynomial Functions:**
    - For any polynomial function n^k, there exists an n_0 such that for all $n > n_0$, $n^{\log n} > n^k$.
    - This is because $\log n$ grows unboundedly as n increases, making $n^{\log n}$ eventually larger than any fixed polynomial n^k.
2. **Compare with Exponential Functions:**
    - For any exponential function a^n, there exists an n_0 such that for all $n > n_0$, $a^n > n^{\log n}$.
    - This is because exponential growth a^n increases at a much faster rate than $n^{\log n}$ as n increases.

Mathematical Intuition:

- **Polynomial Growth (e.g., n^2):** As n increases, the rate of growth is dependent on the power k. For example, doubling n approximately quadruples n^2.
- **Sub-exponential Growth (e.g., $n^{\log n}$):** The rate of growth here is determined by the logarithmic term in the exponent. It grows faster than any polynomial because the exponent itself is growing.
- **Exponential Growth (e.g., 2^n):** The rate of growth here is the fastest among the three, as the base a is raised to the power of n. Doubling n leads to squaring the value of a^n.

Therefore, $f(n) = n^{\log n}$ fits perfectly between polynomial and exponential growth rates, making it a suitable example of a function whose order of growth is larger than any polynomial but smaller than any exponential function.

Here's a Java program that demonstrates the order of growth of the function $f(n) = n^{\log n}$:

```
public class GrowthFunction {

    public static void main(String[] args) {
        int n = 32; // Example value of n
        long startTime = System.nanoTime();

        // Example code block with time complexity proportional to
n^log(n)
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= Math.pow(2, Math.log(i) /
Math.log(2)); j++) {
                // Some constant-time operation
                int x = i + j;
            }
        }

        long endTime = System.nanoTime();
        long duration = (endTime - startTime); // Execution time in
nanoseconds
        System.out.println("Execution time: " + duration + "
nanoseconds");
    }
}
```

Explanation:

- **Nested Loops**: The program has nested loops where the inner loop runs $2^{\log i}$ times for each iteration of the outer loop.
- **Time Complexity**: Each iteration of the inner loop executes a constant-time operation (here represented by `int x = i + j;`), and the number of iterations overall is proportional to $n^{\log n}$.

Output:

- The program measures the execution time using `System.nanoTime()` to get a rough idea of the time taken, but it's important to note that this won't perfectly reflect $n^{\log n}$ due to system-level variations and other factors.

This example demonstrates the growth complexity in terms of operations related to $n^{\log n}$. For precise control over exact execution time reflecting $n^{\log n}$, specialized hardware or simulation environments would be required. This Java program effectively shows the order of growth of $f(n) = n^{\log n}$.